ENGR101: Lecture 2

Computer architecture. Binary numbers.

ECS, VUW

2024

What we cover today?

1. Computer architecture, basic hardware
2. Machine codes
3. Binary numbers
4. Conversion from binary to decimal and back
5. Hexadecimal

Computer architecture 1: What is inside?

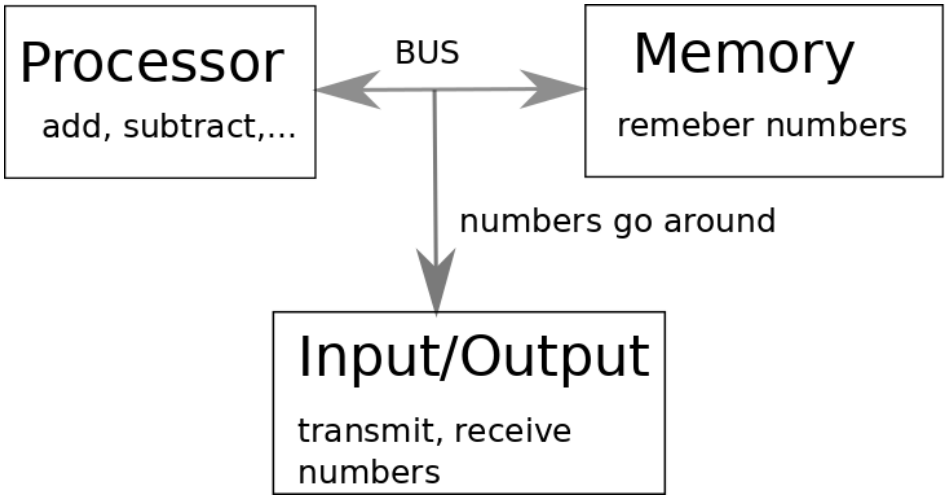
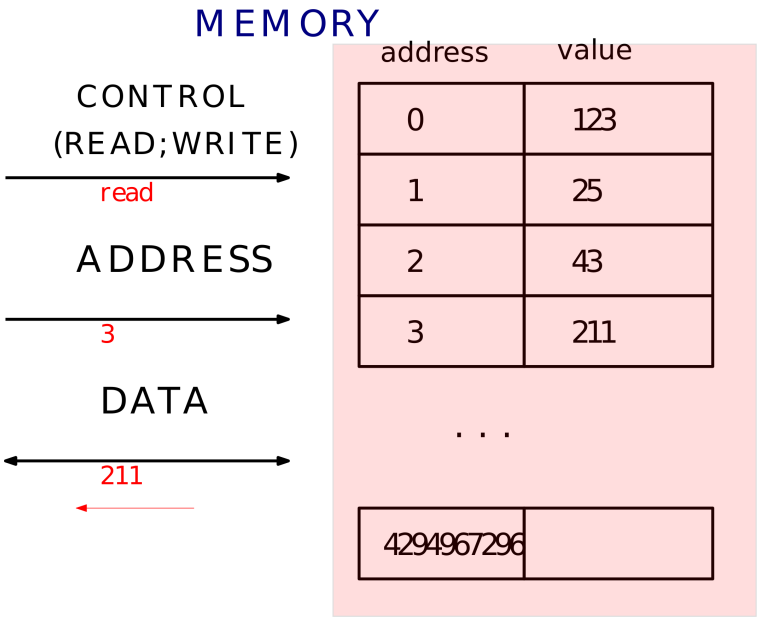


Figure: Computer architecture

- There are three major parts
  1. Processor (CPU, central processing unit)
  2. Memory
  3. Input-output
- All these parts are connected by **buses**. Not this bus

Data(numbers) travel between these parts along the buses.

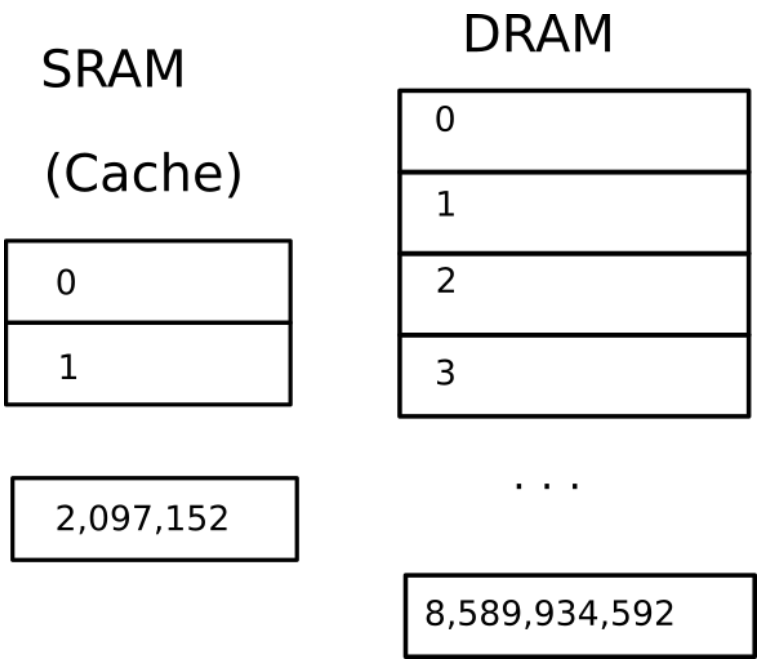
Memory - where all the smarts are



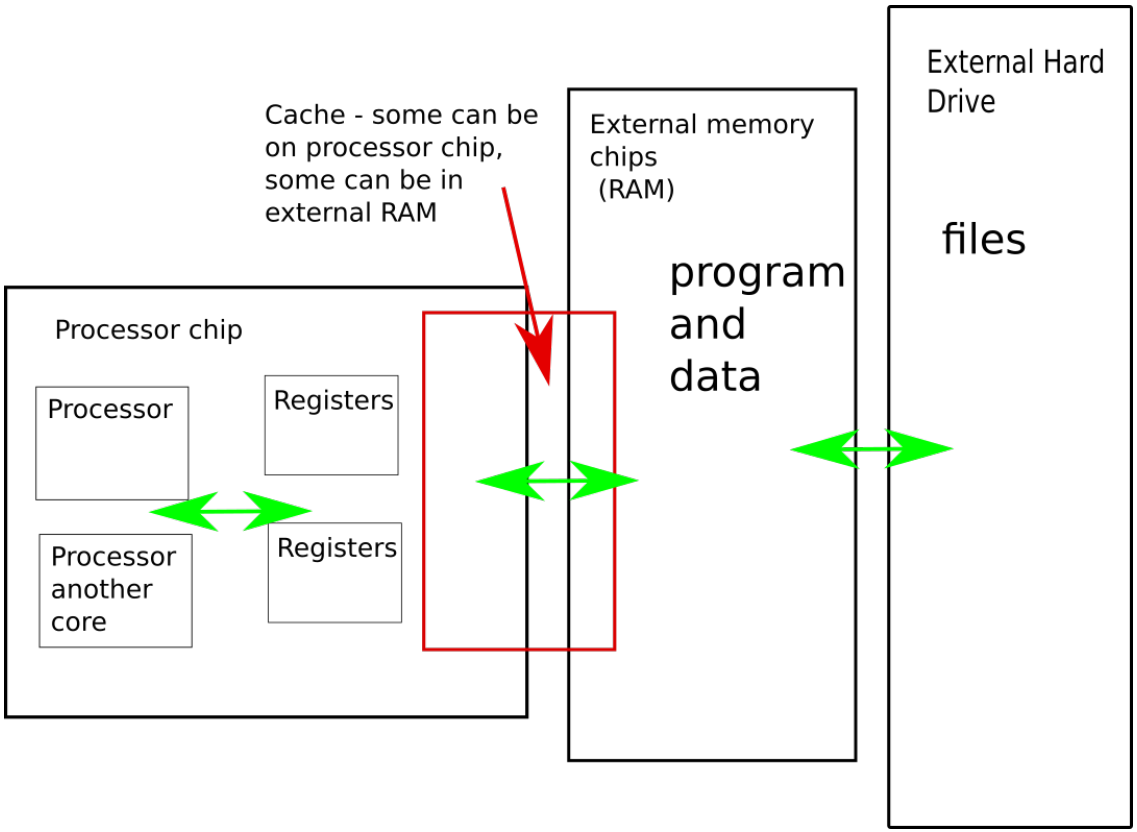
- Set of numbered boxes. Box number called address (like street house number)
- Inside each box there is a number . It is called value.
- Can read value from the box of known address
- Can write value to the box. Old value is gone (overwritten).
- If we want to work with the value - we have to know the address

It is called RAM - random access memory (random - you can read/write using any valid address).

Memory - tradeoff



- There is a trade-off between memory size and speed
- There is hardware which allows fast read/write but it is bulky (Static RAM)
- There is hardware with higher storage density (Dynamic RAM) but it is slow
- To get best of both memory is arranged in layers



- Layers are:
  - registers (super fast)
  - cache (fast)
  - RAM (hmm)
  - Hard Disk (slow as)
- Computer constantly tries to guess which data should be in fast memory (cache) right now

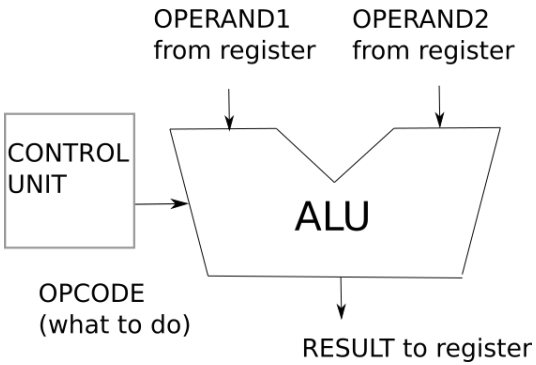
Figure: Memory hardware

Lesson from that - keep your data localized if you want speed (much more on it later).

Processor

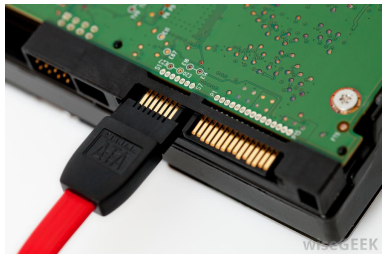
Looks like:

Usually shown as:



- Calculator, basically. But fast. ALU and control unit.
- ALU stands for Arithmetic Logic Unit
- Takes up to two numbers (operands) - from registers
- Does some operation to them: add, subtract, etc.
- Gets the result. Puts it into register.
- Control Unit sets ALU for the operation

Bus



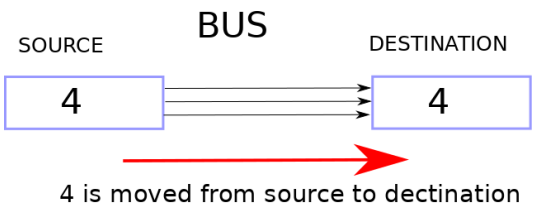


Figure: Moves numbers around

- Bus is bunch of wires and set of rules
- Bus **protocol** determines how numbers are moved from the source to the destination

Input/Output

Keyboard, mouse, camera, files on hard drive, etc. Standard PCs have inputs/outputs which can take/produce digital signals only (i.e. numbers). Sound cards? Sound is not a number... Or is it?

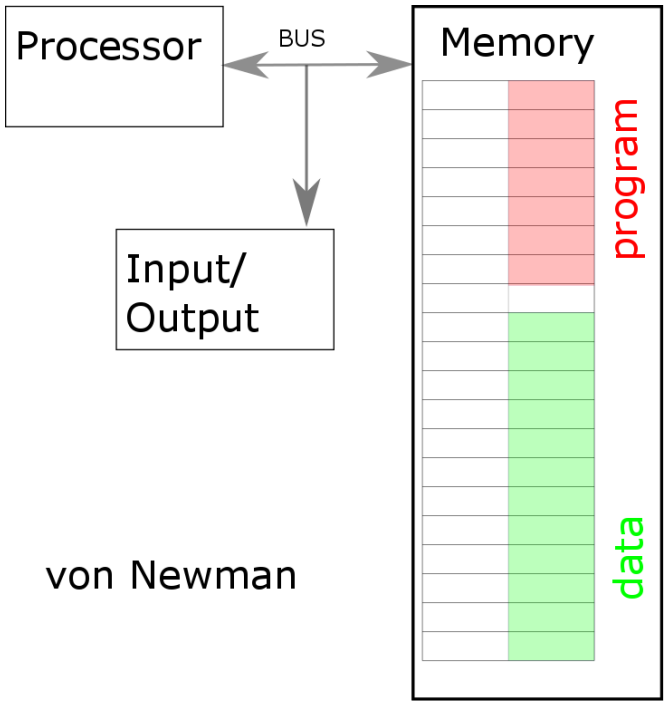
Figure:

It is digital

Looking at very big picture, computers take numbers in, do something with these numbers and produce output (numbers again).
It is numbers all the way down.
Way different from biological wetware.
If everything is stored as numbers how computer knows what to do? How programs are represented as numbers?
Answer is called **encoding** - everybody agreed that number means something else. There is a lot of encoding happening in software.
First we have a look how programs (instructions) are encoded.

Stored program concept - 1

John von Neumann and Alan Turing both proposed the stored program concept in separate publications in 1945.

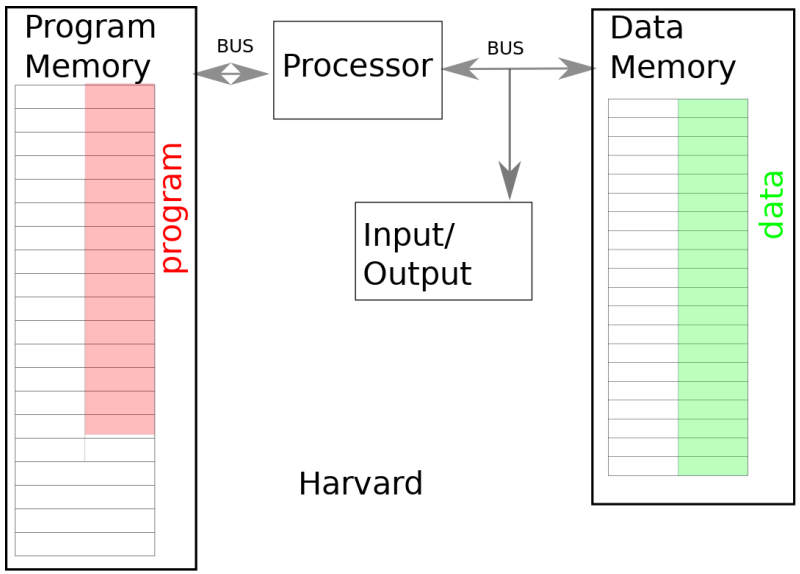


von Newman

Figure:

Program is a sequence of instructions. Instructions for the processor are stored in memory. Each instruction is fetched from main memory one at a time, decoded and executed in the processor. CPU instructions are stored in memory as numbers.
Data also are stored in the memory. Ii it is same memory - then it is called von Newman architecture.

Harvard architecture



- Sometimes different architecture is used - program and data are stored in different memory hardware.
- Program usually can not be modified without physical access to the computer and specialized hardware.

PCs follow von Newman architecture. Software can be modified easily and even remotely.

Program stored as set of numbers

If we managed to open program file we will see something like:
0x 60 00 00 80
0x A4 00 00 00
0x 60 01 00 84
0x A4 01 01 00
0x 60 02 00 00
0x 60 03 00 04
0x 60 04 00 00
Each of these numbers is instruction for processor to do some operation.

Numbers and nothing else. Works in steps.

How to produce this sequence - that is a BIG question.
Machine codes are executed one after another in cycle:

1. Fetch - Retrieve an instruction from the memory.
2. Decode - Translate the retrieved instruction into a series of computer commands.
3. Execute - Execute the computer commands.
    1. can be read or write
    2. can be calculation

Each step of the cycle is done on **clock** tick. Clock is ticking fast - billions of times per second.

Summary so far:

- Numbers and nothing else
- All CPU instructions are stored in memory as numbers too. And first computers were programmed entering these numbers directly.
- Of course, it is hard to remember codes for all instructions and memory addresses. Sometimes programmers have to do that, but now-days it is an exception.
- There are programs which convert human-readable commands into machine codes.

We will look at how to engage these programs during next lecture.
For now we will look at how numbers are represented inside the computer and what are the limitations of this representation.

Numbers: Decimal system

We are used to decimal system - we use 10 digits.
So if we start counting... 1,2,3,4,5,6,7,8,9..
Now we run out of symbols to write the number (10), what we do?
When we want to represent 10 we use next digit to the left:
$10 = 10*1 + 0$.
It is positional system and value in second digit is count of how many 10s there are in this number.

Positional number system

Contribution of the value of every digit depends on which digit it is - system we use is positional.

Digit	digit5	digit4	digit3	digit2	digit1
Weight	$10^4 = 10000$	$1000 = 10^3$	$100 = 10^2$	$10 = 10^1$	$1 = 10^0$

Example: Number 567 can be written as:
$567 = 5 * 100 + 6 * 10 + 7 * 1$
Not all number systems are positional. Roman, for example, is not.

Question: Why we use 10 symbols?

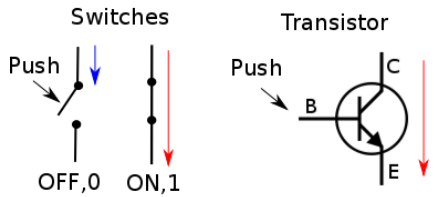
1. We have 10 fingers
2. 10 is magic number

Other systems are possible. What is used in computing? It is determined by hardware.

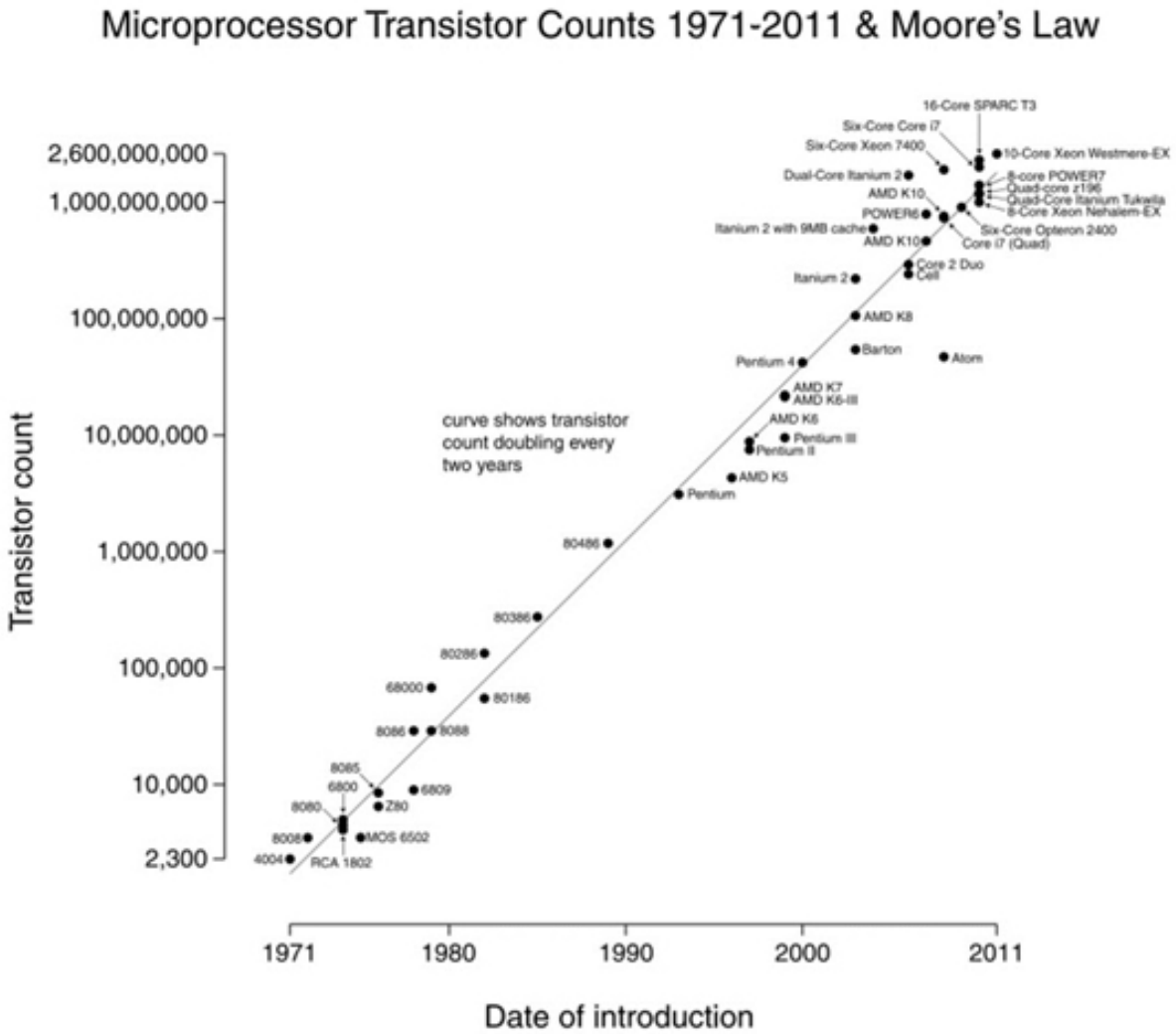
Computing hardware - switches(transistors)

You can build any digital circuit using switches only!

- In computing binary system is used because switches are base of computer hardware. Switch has 2 states: ON and OFF.
- In electronics transistors are used as switching elements
- Transistor currently is 5 nm long. Atom of silicon is 100 pm long. So one transistor is about 100 atoms big.
- How many transistors are in average computer?

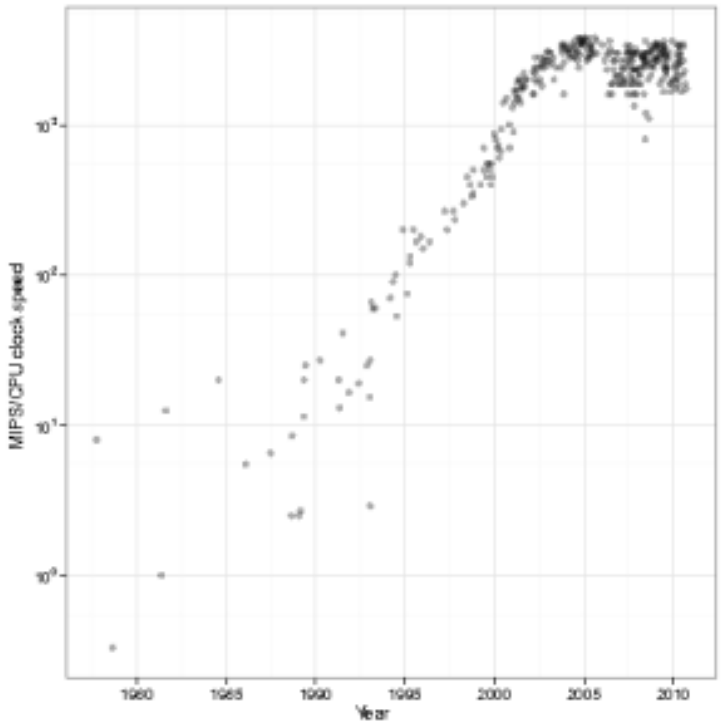


Moore's law (not a law of nature, more of an observation)



How fast can you flip the switch?

Clock rate determines how fast transistor can switch.
Bad news is that there is a limit on clock rate - and we reached it. Speed is stagnant at about 4 GHz (4 billion times per second). Interesting times...



Clock rate is not same as instructions rate.
Each instruction can take several clock cycles. Sometimes a lot more.
Bottleneck usually is exchange with memory - can take up to 70 clock cycles.
It is measured in CPI (cycles per instruction). Computer with good architecture and low clock rate can outperform badly designed processor with high clock rate.

Counting in binary 1 - binary digit weights

In decimal we use 10 symbols for each position.
For decimal system:

Digit	digit5	digit4	digit3	digit2	digit1
Weight	$10^4 = 10000$	$1000 = 10^3$	$100 = 10^2$	$10 = 10^1$	$1 = 10^0$

In binary we use 2 symbols for each position.
For binary system digit weights are:

Digit	digit6	digit5	digit4	digit3	digit2	digit1
Weight	$32 = 2^5$	$16 = 2^4$	$8 = 2^3$	$4 = 2^2$	$2 = 2^1$	$1 = 2^0$

binary to decimal conversion -1

How get value of decimal numbers?
567, for example

- Start from leftmost digit. $\mathbf{5}67$
  Take value of the digit (5). Multiply with weight(100). Remember result -500.
- Move one digit right $5\mathbf{6}7$
  Take value of the digit (6). Multiply with weight(10). Remember result - 60.
- Move one digit right $56\mathbf{7}$
  Take value of the digit (7). Multiply with weight(1). Remember result - 7.
- Add all results together. $567 = 500 + 60 + 7$

binary to decimal conversion -2

Follow same procedure for binary number: 1011_2 ?

Note: We will use subscripts to denote base of the system.

- Start from leftmost digit. $\mathbf{1}011$
  Take value of the digit (1). Multiply with weight of the digit -8. Remember result - 8.
- Move one digit right $1\mathbf{0}11$
  Take value of the digit (0). Multiply with weight of this digit-4. Remember result - 0.
- Move one digit right $10\mathbf{1}1$
  Take value of the digit (1). Multiply with weight - 2. Remember result - 2.
- Move one digit right $101\mathbf{1}$
  Take value of the digit (1). Multiply with weight - 1. Remember result - 1.
- Add all results together. $1011_2 = 8 + 0 + 2 + 1 = 11_{10}$

Digit	digit6	digit5	digit4	digit3	digit2	digit1
Weight	$32 = 2^5$	$16 = 2^4$	$8 = 2^3$	$4 = 2^2$	$2 = 2^1$	$1 = 2^0$

Decimal to binary - 1

We use this table:

Digit	digit6	digit5	digit4	digit3	digit2	digit1
Weight	$32 = 2^5$	$16 = 2^4$	$8 = 2^3$	$4 = 2^2$	$2 = 2^1$	$1 = 2^0$

but do it backward.

For example: we want binary representation of 25. We start with XXXXXX - and replace X with 1 or 0

- Look at the table
- How many "32"s are in 25 - none. Here goes - 0XXXXX
- How mant "16"s are in 25 - one. It becomes - 01XXXX (which is at least 16, can be bigger)
- What remains? 25 - 16 = 9

Decimal to binary - 2

- Now we have 01XXXX and 9 to represent in XXXX
- How many "8" in - one. Put 1 into position for 8 : 011XXX
- Now we have 011XXX which is at least 16+8=24, but can be bigger
- 1 remains. There is no 4 in 1 : 0110XX
- 1 still remains. We can not fit 2 into 1. Here goes: 01100X
- 1 still remains. And we can fit it. Here goes: 011001

$25_{10} = 011001_2$

Question time!

- Convert 0010 to decimal:
  - A: 1
  - B: 2
  - C: 8
- Convert 1010 to decimal:
  - A: 2
  - B: 8
  - C: 10

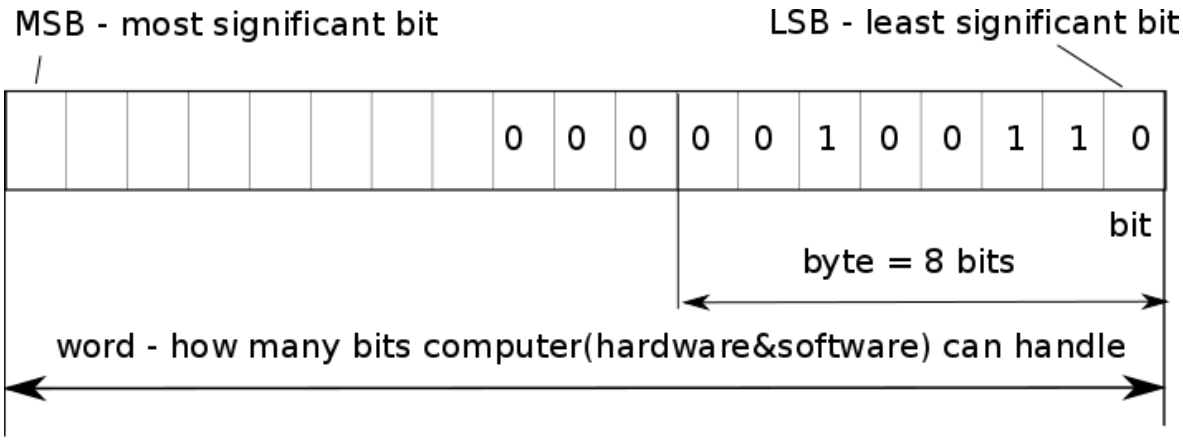
Question time - carry on!

Digit	digit6	digit5	digit4	digit3	digit2	digit1
Weight	$32 = 2^5$	$16 = 2^4$	$8 = 2^3$	$4 = 2^2$	$2 = 2^1$	$1 = 2^0$

- Convert 7 to binary:
  - A: 0111
  - B: 1100
  - C: 1001
- Convert 2 to binary:
  - A: 0010
  - B: 0011
  - C: 1011

Bits, bytes, words

Every digit of a binary number is called bit (BInary digiT).



Intel C4004 microprocessor,Intel,1971 - 4 bits word
Intel 8008,1972 - 8 bits word.
Intel 8086,1976 - 16 bits word (PC original design).
Now most PCs use 64 bit processors. Software moving from 32 bits to 64.

Hexadecimal

Binary is an underlying type for any digital circuits, computers included.
But to write and read numbers in binary notation can be tedious.
There is an another notation which makes it easier and so is used quite often.
Hexadecimal: In hexadecimal base 16 is used. So we start counting 0 1 2 3 4 5 6 7 8 9... Oops, we run out of symbols.
Letters are used: A instead of 10
B instead of 12
C instead of 13
D instead of 14
E instead of 15
F instead of 16

Digit	digit5	digit4	digit3	digit2	digit1
Weight	$65536 = 16^4$	$4066 = 16^3$	$256 = 16^2$	$16 = 16^1$	$1 = 16^0$

We can squeeze bigger value in same number of digits.

Decimal	Binary	Hex
0	0000 0000	0
1	0000 0001	1
2	0000 0010	2
10	0000 1010	A
11	0000 1011	B
15	0000 1111	F
16	0001 0000	10
254	1111 1110	FE
255	1111 1111	FF

- Two hexadecimal digits correspond exactly to one byte, unlike the strange 255 limit in decimal, the hexadecimal byte limit is FF.
- In programming languages, a hexadecimal number is usually prefixed with '0x' to make the compiler aware that you are using hexadecimal.

Question

Digit	digit5	digit4	digit3	digit2	digit1
Weight	$65536 = 16^4$	$4066 = 16^3$	$256 = 16^2$	$16 = 16^1$	$1 = 16^0$

Convert hexadecimal 0x11 to Decimal:

- A: 16 (decimal)
- B: 17 (decimal)
- C: 11 (decimal)